WYŻSZA SZKOŁA
INFORMATYKI i ZARZĄDZANIA
z siedzibą w Rzeszowie

KOLEGIUM INFORMATYKI STOSOWANEJ

Kierunek: INFORMATYKA

Specjalność: Programowanie

Dominik Duda
Nr albumu studenta w69775

Elektroniczny system oceniania

Prowadzący: mgr inż. Ewa Żesławska

Praca projektowa programowanie obiektowe C#

Rzeszów 2025

Spis treści

Wstęp

Współczesne systemy edukacyjne coraz częściej korzystają z narzędzi cyfrowych, które mają na celu usprawnienie procesu nauczania oraz oceniania uczniów. Tradycyjne metody sprawdzania wiedzy wymagają dużego nakładu pracy ze strony nauczycieli – ręczne sprawdzanie testów, ocenianie odpowiedzi czy zarządzanie wynikami zajmuje znaczącą ilość czasu, który mógłby zostać wykorzystany na inne aspekty dydaktyczne.

W odpowiedzi na ten problem powstał Elektroniczny System Oceniania – innowacyjna platforma umożliwiająca automatyzację procesu oceniania. System pozwala nauczycielom na tworzenie testów, dodawanie użytkowników do wirtualnych klas oraz automatyczne ocenianie ich wyników, eliminując potrzebę ręcznej ingerencji w proces sprawdzania prac. Dzięki temu nauczyciele mogą skupić się na przekazywaniu wiedzy, a uczniowie otrzymują natychmiastową informację zwrotną na temat swoich wyników.

Automatyzacja oceniania nie tylko przyspiesza proces edukacyjny, ale także minimalizuje ryzyko błędów ludzkich i zwiększa obiektywność ocen. System ten jest więc krokiem w stronę nowoczesnej edukacji, w której technologia wspiera zarówno nauczycieli, jak i uczniów, tworząc bardziej efektywne i sprawiedliwe środowisko nauki.

Rozdział 1

Opis założeń projektu

1.1 Cele projetu

Celem projektu jest stworzenie Elektronicznego Systemu Oceniania, który pozwala na automatyzację procesu oceniania testów oraz zarządzanie wynikami uczniów. Tradycyjne metody oceniania wymagają dużego nakładu pracy nauczycieli, którzy muszą ręcznie sprawdzać testy, analizować wyniki i przekazywać informacje zwrotne uczniom. System ten ma na celu usprawnienie tego procesu, eliminując konieczność ręcznego oceniania oraz umożliwiając nauczycielom skoncentrowanie się na przekazywaniu wiedzy i rozwijaniu umiejętności uczniów.

Problemem, który zostanie rozwiązany, jest długi czas potrzebny na ocenianie testów oraz subiektywność ocen. Główną przyczyną tego problemu jest konieczność ręcznego sprawdzania prac przez nauczycieli, co pochłania znaczną część ich czasu. Dodatkowo, różnice w interpretacji odpowiedzi mogą prowadzić do niespójności ocen, co wpływa na uczniów i ich poczucie sprawiedliwości.

Problem ten jest istotny, ponieważ wpływa na efektywność procesu edukacyjnego. Badania pokazują, że nauczyciele poświęcają znaczną część swojego czasu na sprawdzanie testów, co ogranicza ich możliwości indywidualnego podejścia do uczniów. Ponadto, subiektywność ocen może powodować konflikty oraz wpływać na motywację uczniów. Automatyzacja oceniania przyczyni się do zwiększenia przejrzystości procesu oraz zmniejszenia obciążenia nauczycieli.

Aby problem został skutecznie rozwiązany, konieczne jest zaprojektowanie i wdrożenie systemu umożliwiającego automatyczne ocenianie testów, przechowywanie wyników oraz generowanie raportów. System powinien być intuicyjny i dostępny dla nauczycieli oraz uczniów, a także zapewniać możliwość dostosowywania testów do różnych przedmiotów i poziomów trudności.

Realizacja projektu została podzielona na kilka etapów, które obejmowały stopniowe dodawanie funkcjonalności systemu. Na początku stworzono panel logowania, który stanowił podstawę uwierzytelniania użytkowników. Następnie wdrożono panel administratora, umożliwiający dodawanie nauczycieli i uczniów do systemu.

Kolejnym krokiem była implementacja panelu użytkownika dla nauczycieli. W tej części systemu nauczyciele uzyskali możliwość tworzenia klas, dodawania testów oraz przeglądania ocen uczniów. Po zakończeniu tej fazy skoncentrowano się na funkcjonalności dla uczniów. Stworzono panel ucznia, w którym użytkownicy mogą sprawdzać, do jakich klas należą, jakie testy są dla nich dostępne oraz jakie oceny otrzymali.

Dzięki takiemu podejściu system został rozwijany w sposób logiczny i umożliwił stopniowe testowanie oraz optymalizację kolejnych funkcjonalności. Ostatecznym wynikiem prac jest w pełni funkcjonalna aplikacja wspierająca proces oceniania uczniów w sposób zautomatyzowany i przejrzysty.

1.2 Wymagania funkcjonalne i niefunkcjonalne

Wymagania systemowe można podzielić na funkcjonalne i niefunkcjonalne. Wymagania funkcjonalne określają, jakie operacje i usługi powinien wykonywać system, natomiast wymagania niefunkcjonalne odnoszą się do jego charakterystyk, takich jak wydajność, bezpieczeństwo czy łatwość obsługi.

Wymagania funkcjonalne obejmują możliwość tworzenia testów przez nauczycieli, automatyczne sprawdzanie odpowiedzi, generowanie raportów z wynikami oraz zarządzanie użytkownikami. System umożliwia dodawanie pytań zamkniętych jednokrotnego wyboru z różną ilością odpowiedzi. Ponadto, system zapewnia możliwość analizy wyników uczniów oraz ich postępów w czasie.

Wymagania niefunkcjonalne dotyczą aspektów takich jak bezpieczeństwo danych, niezawodność i skalowalność systemu. System powinien działać w środowisku sieciowym, zapewniać ochronę danych użytkowników oraz możliwość integracji z innymi platformami edukacyjnymi w przyszłości. Oczekuje się również, że interfejs użytkownika będzie intuicyjny i przyjazny dla nauczycieli oraz uczniów, co ułatwi jego wdrożenie i użytkowanie w związku z czym należy dorobić gui w celu łatwiejszej interakcji z użytkownikiem.

System spełnia wymagania dotyczące wydajności(mniej niż 500ms na odpowiedź), umożliwiając szybkie generowanie raportów oraz ocenianie testów w czasie rzeczywistym. Dodatkowo, konieczne jest zapewnienie wysokiego poziomu dostępności oraz minimalizacja ryzyka błędów i awarii, co pozwoli na niezawodne funkcjonowanie systemu w warunkach szkolnych. Aplikacja póki co jest nie zoptymalizowana do dużej ilości zapytań i nie działa poprawnie w środowiku wielowątkowym w związku z czym nie jest gotowa na wystawienie jej w środowisku produkcyjnym. Jeśli chodzi o zarządzanie wyjątkami i odpornośc na błędy trzeba jeszcze poprawić kilka aspektów. Aplikacja nie zapewnia też wystarczającego poziomu bezpieczeństa wymaganego w środowiku szkolnym

Rozdział 2

Opis struktury projektu

W niniejszym rozdziale przedstawiona zostanie struktura projektu, wraz z opisem technicznym oraz wykorzystanymi narzędziami.

2.1 Technologie i narzędzia

Projekt został zaimplementowany w języku C# bez użycia dodatkowego frameworka. Do zarządzania bazą danych wykorzystano Microsoft SQL Server, a połączenie realizowane jest za pomocą biblioteki Microsoft.Sql. Interfejs aplikacji opiera się na konsoli, co oznacza, że nie posiada ona klasycznego frontendu.

Minimalne wymagania sprzętowe dla aplikacji to:

- Procesor: dowolny zgodny z wymaganiami .NET,
- Pamięć RAM: 2GB,
- Wolne miejsce na dysku: 100MB,
- System operacyjny: Windows 10 lub nowszy.

2.2 Struktura bazy danych

Baza danych została utworzona i zarządzana przy pomocy SQL Server Management Studio 20. Cała struktura bazodanowa znajduje się w schemacie `dbo`. Nie zastosowano zdefiniowanych relacji, a łączenia odbywają się za pomocą prostych zapytań `JOIN`. Brak w systemie zaawansowanych zapytań.

2.3 Hierarchia klas

W projekcie zastosowano głównie wzorzec projektowy strategia. Cała aplikacja jest kontrolowana z klasy `MainProgram`, gdzie pętla programu kończy się po wpisaniu wartości `-1`. Przechodzenie między interfejsami realizowane jest przez metody statyczne w tej klasie.

2.4 Opis klasy MainProgram

Klasa `MainProgram` pełni rolę głównego punktu wejścia do aplikacji. Odpowiada za inicjalizację repozytoriów, interfejsów użytkownika oraz zarządzanie przepływem sterowania między różnymi panelami systemu.

2.4.1 Atrybuty statyczne

- **`databaseManager`** – obiekt klasy `DatabaseManager`, odpowiedzialny za zarządzanie połączeniem z bazą danych.
- **`authRepository`** – repozytorium odpowiedzialne za uwierzytelnianie użytkowników.
- **`studentRepository`** – repozytorium obsługujące operacje na encjach uczniów.
- **`teacherRepository`** – repozytorium obsługujące operacje na encjach nauczycieli.
- **`loginInterface`** – interfejs obsługujący logowanie użytkownika.
- **`adminInterface`** – interfejs administratora, umożliwiający zarządzanie użytkownikami.
- **`studentsClassRepository`** – repozytorium obsługujące operacje na klasach uczniów.
- **`questionRepository`** – repozytorium obsługujące operacje na pytaniach testowych.
- **`testAtemptRepository`** – repozytorium obsługujące podejścia uczniów do testów.
- **`answearRepository`** – repozytorium obsługujące operacje na odpowiedziach uczniów.
- **`testRepository`** – repozytorium obsługujące operacje na testach.

2.4.2 Metody statyczne

`Main(string[] args)`

Opis: Jest to główna metoda aplikacji, odpowiedzialna za uruchomienie systemu.
Działanie:

1. Przechwytuje potencjalne wyjątki i wyświetla ich treść na konsoli.
2. Uruchamia interfejs logowania (`loginInterface.handleChoosedOperation()`).

`switchToAdminPanel()`

Opis: Metoda umożliwia przełączenie do panelu administratora.
Działanie:

1. Tworzy nieskończoną pętlę `while (true)`, aby administrator mógł wykonywać wiele operacji.
2. Obsługuje potencjalne wyjątki.
3. Przekazuje sterowanie do interfejsu administratora (`adminInterface.handleChoosedOperation`

`switchToStudentPanel(Student student)`

Opis: Metoda umożliwia przełączenie do panelu ucznia.
Parametry: `Student student` – obiekt ucznia, dla którego uruchamiany jest panel.
Działanie:

1. Tworzy obiekt `StudentUserInterface`, przekazując do niego odpowiednie repozytoria oraz instancję ucznia.
2. Uruchamia nieskończoną pętlę, w której wywoływana jest metoda `handleChoosedOperation()`.
3. Obsługuje ewentualne wyjątki.

`switchToTeacherPanel(Teacher teacher)`

Opis: Metoda umożliwia przełączenie do panelu nauczyciela.
Parametry: `Teacher teacher` – obiekt nauczyciela, dla którego uruchamiany jest panel.
Działanie:

1. Tworzy obiekt `TeacherUserInterface`, przekazując do niego odpowiednie repozytoria oraz instancję nauczyciela.
2. Uruchamia nieskończoną pętlę, w której wywoływana jest metoda `handleChoosedOperation()`.
3. Obsługuje ewentualne wyjątki.

2.4.3 Podsumowanie

Klasa `MainProgram` działa jako centralny punkt aplikacji. Steruje przepływem użytkownika pomiędzy różnymi interfejsami (logowania, administratora, ucznia, nauczyciela). Obsługuje również potencjalne wyjątki, minimalizując możliwość niekontrolowanego przerwania działania programu.

2.5 Opis klasy Credidential

Klasa `Credidential` reprezentuje dane uwierzytelniające użytkownika.

2.5.1 Atrybuty

- **`login`** – nazwa użytkownika w postaci ciągu znaków (`string`).
- **`password`** – hasło użytkownika w postaci ciągu znaków (`string`).

2.5.2 Metody

`Credidential(string login, string password)`

Opis: Konstruktor klasy, który inicjalizuje dane logowania użytkownika.
Parametry:

- `login` – login użytkownika.
- `password` – hasło użytkownika.

2.6 Opis klasy LoginInterface

Klasa `LoginInterface` odpowiada za interakcję z użytkownikiem podczas logowania oraz przekierowanie do odpowiedniego panelu systemu.

2.6.1 Atrybuty

- **`operations`** – lista operacji dostępnych dla użytkownika.
- **`authRepository`** – repozytorium obsługujące uwierzytelnianie.
- **`studentRepository`** – repozytorium przechowujące dane uczniów.
- **`teacherRepository`** – repozytorium przechowujące dane nauczycieli.

2.6.2 Metody

`LoginInterface(AuthRepository authRepository, StudentRepository studentRepo TeacherRepository teacherRepository)`

Opis: Konstruktor inicjalizujący repozytoria wykorzystywane w procesie logowania.
Parametry:

- `authRepository` – repozytorium obsługujące logowanie.
- `studentRepository` – repozytorium uczniów.
- `teacherRepository` – repozytorium nauczycieli.

`handleLoginAsStudent()`

Opis: Obsługuje proces logowania ucznia.
Działanie:

1. Pobiera dane logowania od użytkownika.
2. Weryfikuje poprawność danych.
3. Jeśli logowanie zakończy się sukcesem, przekazuje sterowanie do panelu ucznia.

`handleLoginAsTeacher()`

Opis: Obsługuje proces logowania nauczyciela.
Działanie:

1. Pobiera dane logowania od użytkownika.
2. Weryfikuje poprawność danych.
3. Jeśli logowanie zakończy się sukcesem, przekazuje sterowanie do panelu nauczyciela.

`handleLoginAsAdmin()`

Opis: Obsługuje proces logowania administratora.
Działanie:

1. Pobiera dane logowania od użytkownika.
2. Weryfikuje poprawność danych.
3. Jeśli logowanie zakończy się sukcesem, przekazuje sterowanie do panelu administratora.

`authenticateStudent(string login, string password)`

Opis: Metoda uwierzytelnia ucznia na podstawie podanych danych logowania.
Działanie:

1. Pobiera encję logowania z repozytorium.
2. Sprawdza zgodność loginu i hasła.
3. Jeśli dane są poprawne, pobiera obiekt ucznia i zwraca go.

`authenticateTeacher(string login, string password)`

Opis: Metoda uwierzytelnia nauczyciela na podstawie podanych danych logowania.
Działanie:

1. Pobiera encję logowania z repozytorium.
2. Sprawdza zgodność loginu i hasła.
3. Jeśli dane są poprawne, pobiera obiekt nauczyciela i zwraca go.

`authenticateAdmin(string login, string password)`

Opis: Metoda uwierzytelnia administratora na podstawie podanych danych logowania.
Działanie:

1. Pobiera encję logowania administratora.
2. Sprawdza zgodność loginu i hasła.
3. Jeśli dane są poprawne, zwraca wartość `true`.

`authenticate(string loginFromUser, string passwordFromUser, Login login)`

Opis: Pomocnicza metoda walidująca dane logowania.
Działanie:

1. Sprawdza, czy podany login istnieje.
2. Jeśli login jest niepoprawny, wypisuje komunikat o błędzie.
3. Sprawdza, czy hasło jest poprawne.
4. Jeśli dane są poprawne, zwraca `true`, w przeciwnym razie `false`.

`getCredidential()`

Opis: Pobiera od użytkownika login i hasło.
Działanie:

1. Pyta użytkownika o login.
2. Pyta użytkownika o hasło.
3. Zwraca obiekt `Credidential` z wprowadzonymi danymi.

`handleChoosedOperation()`

Opis: Obsługuje wybór użytkownika na ekranie logowania.
Działanie:

1. Wyświetla listę dostępnych operacji.
2. Pobiera wybór użytkownika.
3. Wykonuje odpowiednią akcję na podstawie wyboru.
4. W przypadku wybrania opcji wyjścia kończy działanie programu.

2.6.3 Podsumowanie

Klasa `LoginInterface` umożliwia użytkownikowi logowanie jako uczeń, nauczyciel lub administrator. Weryfikuje dane logowania i przekierowuje użytkownika do odpowiedniego panelu. Obsługuje błędne dane wejściowe, informując użytkownika o nieprawidłowościach.

2.7 Opis klasy AdminInterface

Klasa `AdminInterface` jest odpowiedzialna za obsługę operacji administratora, takich jak zarządzanie użytkownikami systemu (dodawanie, aktualizowanie, usuwanie oraz przeglądanie listy uczniów i nauczycieli).

2.7.1 Atrybuty

- **`studentRepository`** – repozytorium przechowujące dane uczniów.
- **`teacherRepository`** – repozytorium przechowujące dane nauczycieli.
- **`authRepository`** – repozytorium obsługujące uwierzytelnianie.
- **`operations`** – lista dostępnych operacji administratora.

2.7.2 Metody

`AdminInterface(StudentRepository studentRepository, TeacherRepository teach`
`AuthRepository authRepository)`

Opis: Konstruktor inicjalizujący repozytoria wykorzystywane w zarządzaniu użytkownikami.
Parametry:

- `studentRepository` – repozytorium uczniów.
- `teacherRepository` – repozytorium nauczycieli.
- `authRepository` – repozytorium obsługujące logowanie.

`handleChoosedOperation()`

Opis: Obsługuje wybór użytkownika w interfejsie administratora.
Działanie:

1. Wyświetla dostępne operacje.
2. Pobiera wybór użytkownika.
3. Wykonuje odpowiednią akcję, np. dodawanie, aktualizację lub usuwanie użytkowników.

`handleAddStudent()`

Opis: Obsługuje dodawanie nowego ucznia do systemu.
Działanie:

1. Pobiera dane logowania ucznia.
2. Tworzy nowy obiekt ucznia.
3. Dodaje ucznia do bazy danych za pomocą repozytorium.

`handleAddTeacher()`

Opis: Obsługuje dodawanie nowego nauczyciela do systemu.
Działanie:

1. Pobiera dane logowania nauczyciela.
2. Tworzy nowy obiekt nauczyciela.
3. Dodaje nauczyciela do bazy danych za pomocą repozytorium.

`handleUpdateStudent()`

Opis: Obsługuje aktualizację danych ucznia.
Działanie:

1. Pobiera identyfikator ucznia do aktualizacji.
2. Znajduje ucznia w bazie danych.
3. Pobiera nowe dane ucznia.
4. Aktualizuje dane ucznia w bazie.

`handleUpdateTeacher()`

Opis: Obsługuje aktualizację danych nauczyciela.
Działanie:

1. Pobiera identyfikator nauczyciela do aktualizacji.
2. Znajduje nauczyciela w bazie danych.
3. Pobiera nowe dane nauczyciela.
4. Aktualizuje dane nauczyciela w bazie.

`handleDeleteStudent()`

Opis: Obsługuje usunięcie ucznia z systemu.
Działanie:

1. Pobiera identyfikator ucznia do usunięcia.
2. Usuwa ucznia z bazy danych.

`handleDeleteTeacher()`

Opis: Obsługuje usunięcie nauczyciela z systemu.
Działanie:

1. Pobiera identyfikator nauczyciela do usunięcia.
2. Usuwa nauczyciela z bazy danych.

`handleViewAllStudents()`

Opis: Obsługuje wyświetlenie wszystkich uczniów zapisanych w systemie.
Działanie:

1. Pobiera listę uczniów z repozytorium.
2. Wyświetla listę uczniów na ekranie.

`handleViewAllTeachers()`

Opis: Obsługuje wyświetlenie wszystkich nauczycieli zapisanych w systemie.
Działanie:

1. Pobiera listę nauczycieli z repozytorium.
2. Wyświetla listę nauczycieli na ekranie.

2.7.3 Podsumowanie

Klasa `AdminInterface` umożliwia administratorowi zarządzanie użytkownikami systemu, w tym dodawanie, aktualizowanie, usuwanie oraz przeglądanie listy uczniów i nauczycieli. Zapewnia przejrzystą interakcję z systemem poprzez konsolę.

2.8 Opis klasy StudentUserInterface

Klasa `StudentUserInterface` odpowiada za interakcję ucznia z systemem testów. Pozwala przeglądać dostępne testy, sprawdzać wyniki oraz brać udział w testach.

2.8.1 Atrybuty

- **`testRepository`** – repozytorium przechowujące testy.
- **`testAtemptRepository`** – repozytorium przechowujące podejścia ucznia do testów.
- **`answearRepository`** – repozytorium odpowiedzi ucznia.
- **`student`** – obiekt reprezentujący zalogowanego ucznia.
- **`operations`** – lista operacji dostępnych dla ucznia.

2.8.2 Metody

`StudentUserInterface(TestRepository testRepository, TestAtemptRepository testAtemptRepository, AnswearRepository answearRepository, Student student)`

Opis: Konstruktor inicjalizujący obiekt interfejsu użytkownika dla ucznia.
Parametry:

- `testRepository` – repozytorium testów.
- `testAtemptRepository` – repozytorium podejść do testów.
- `answearRepository` – repozytorium odpowiedzi.
- `student` – obiekt reprezentujący ucznia.

`handleShowTestsToDo()`

Opis: Wyświetla listę testów, których uczeń jeszcze nie wykonał.
Działanie:

1. Pobiera listę nieukończonych testów dla zalogowanego ucznia.
2. Drukuje informacje o dostępnych testach.

`handleShowTestsResults()`

Opis: Wyświetla wyniki ucznia z zakończonych testów.
Działanie:

1. Pobiera listę podejść ucznia do testów.
2. Drukuje wyniki testów dla ucznia.

`handleAttendTest()`

Opis: Umożliwia uczniowi przystąpienie do testu.
Działanie:

1. Pobiera od użytkownika identyfikator testu, do którego chce przystąpić.
2. Sprawdza, czy test o podanym ID istnieje.
3. Jeśli test istnieje, rozpoczyna jego wypełnianie i zapisuje wyniki w bazie danych.

`handleChoosedOperation()`

Opis: Obsługuje wybór operacji w interfejsie ucznia.
Działanie:

1. Wyświetla dostępne operacje.
2. Pobiera wybór ucznia.
3. Wykonuje odpowiednią operację w zależności od wybranej opcji.

2.8.3 Podsumowanie

Klasa `StudentUserInterface` zapewnia uczniowi dostęp do testów, umożliwia przeglądanie dostępnych testów, sprawdzanie wyników oraz przystępowanie do testów. Umożliwia interakcję z systemem poprzez konsolę.

2.9 Opis klasy TeacherUserInterface

Klasa `TeacherUserInterface` umożliwia nauczycielowi interakcję z systemem testów. Pozwala na zarządzanie klasami, testami oraz wynikami uczniów.

2.9.1 Atrybuty

- **`teacher`** – obiekt reprezentujący zalogowanego nauczyciela.
- **`studentRepository`** – repozytorium przechowujące dane uczniów.
- **`studentsClassRepository`** – repozytorium klas uczniów.
- **`questionRepository`** – repozytorium pytań testowych.
- **`answerRepository`** – repozytorium odpowiedzi do testów.
- **`testRepository`** – repozytorium testów.
- **`atemptRepository`** – repozytorium podejść do testów.
- **`operations`** – lista dostępnych operacji dla nauczyciela.

2.9.2 Metody

`TeacherUserInterface(Teacher teacher, StudentRepository studentRepository, StudentsClassRepository studentsClassRepository, QuestionRepository questio AnswearRepository answerRepository, TestRepository testRepository, TestAtem atemptRepository)`

Opis: Konstruktor inicjalizujący interfejs użytkownika nauczyciela.
Parametry:

- `teacher` – obiekt nauczyciela.
- `studentRepository` – repozytorium uczniów.
- `studentsClassRepository` – repozytorium klas.
- `questionRepository` – repozytorium pytań.
- `answerRepository` – repozytorium odpowiedzi.
- `testRepository` – repozytorium testów.
- `atemptRepository` – repozytorium podejść do testów.

`handleChoosedOperation()`

Opis: Obsługuje wybór operacji przez nauczyciela.
Działanie:

1. Wyświetla dostępne operacje.
2. Pobiera wybór nauczyciela.
3. Wykonuje odpowiednią operację w zależności od wyboru.

`handleAddClass()`

Opis: Dodaje nową klasę uczniów i przypisuje do niej nauczyciela.

`handleAddStudentToClass()`

Opis: Dodaje ucznia do istniejącej klasy.
Działanie:

1. Pobiera identyfikator klasy od nauczyciela.
2. Pobiera identyfikator ucznia.
3. Przypisuje ucznia do danej klasy.

`handleViewTestResultByTestId()`

Opis: Wyświetla wyniki uczniów dla konkretnego testu.

`handleAddTest()`

Opis: Tworzy nowy test i przypisuje go do klasy oraz zapisuje powiązane pytania i odpowiedzi.

`handleViewTests()`

Opis: Wyświetla listę testów stworzonych przez nauczyciela.

`handleViewTestById()`

Opis: Wyświetla szczegółowe informacje o teście na podstawie jego identyfikatora.

`handleViewAllStudents()`

Opis: Wyświetla listę wszystkich uczniów w systemie.

`handleViewAllClassForTeacher()`

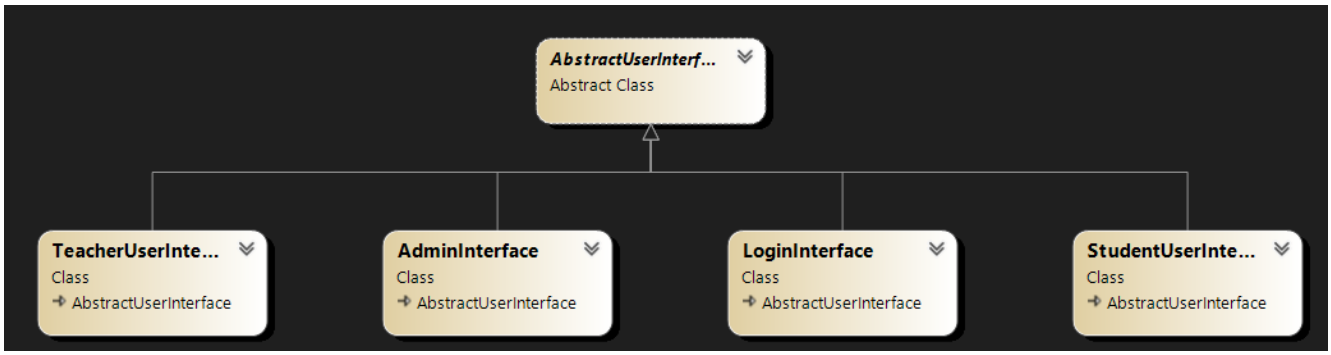
Opis: Wyświetla listę wszystkich klas przypisanych do nauczyciela.

2.9.3 Podsumowanie

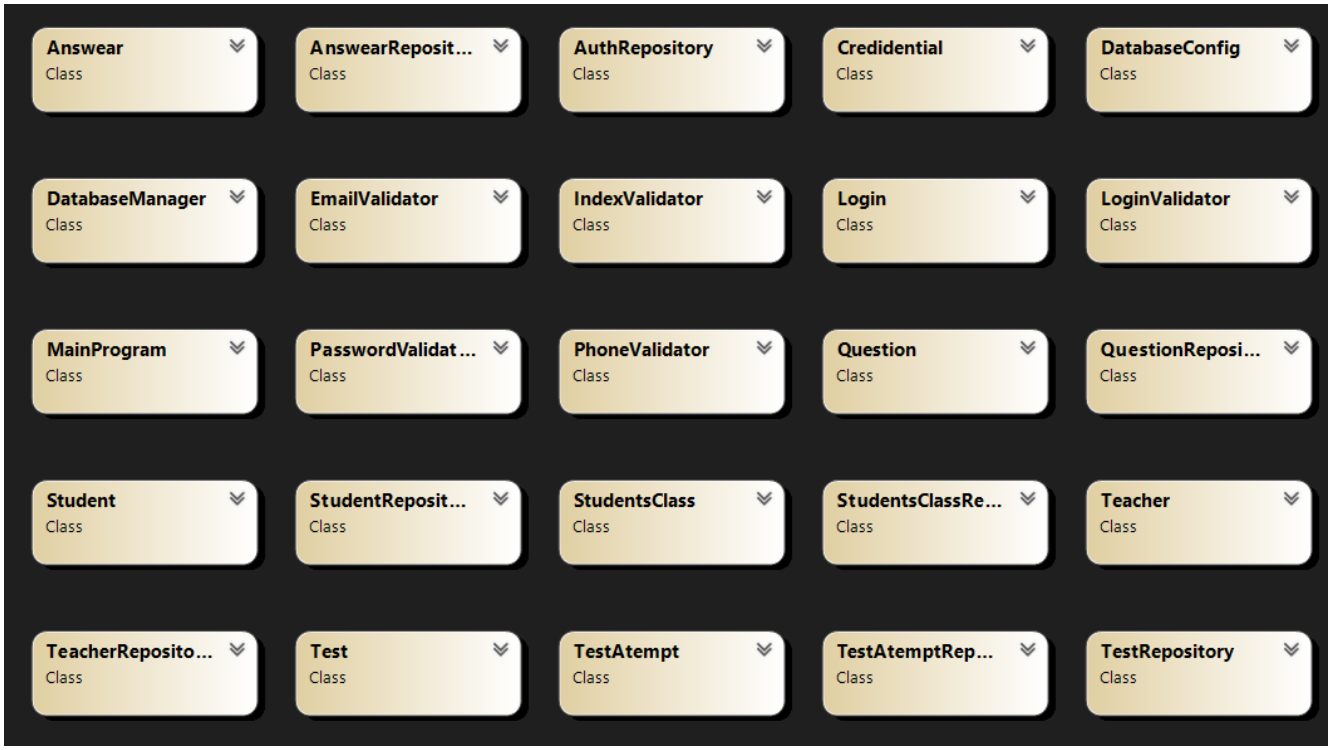
Klasa `TeacherUserInterface` umożliwia nauczycielowi pełne zarządzanie systemem testów, w tym tworzenie klas, dodawanie uczniów, tworzenie testów oraz przeglądanie wyników.

2.10 Diagram klas

Cały diagram znajduje się w plikach źródłowych w repozytorium.Poniżej znajduje się skrócona wersja diagramu klas.



Rysunek 2.1: Diagram klas systemu Rys.1



Rysunek 2.2: Diagram klas systemu Rys.2

Struktura aplikacji obejmuje:

- Klasy rozszerzające implementujące abstrakcyjny interfejs użytkownika, co umożliwia stosowanie wzorca strategii.
- Repozytoria odpowiedzialne za zarządzanie danymi.
- Walidatory dbające o poprawność danych dostarczanych przez użytkownika.
- Encje zawierające metody pomocnicze(przy dalszym rozwoju aplikacji należy rozważyć dodanie metod klas helper ułatwiających procesem wytwórczym kodu)

W kodzie źródłowym znajduje się plik Main.sql umożliwiający szybkie utworzenie struktury bazy danych. W przyszłości można dodać tam także inserty aby były jakieś dane testowe. W aplikacji brakuje warstwy domeny – wszystkie operacje logiczne są obsługiwane przez repozytoria lub interfejsy użytkownika. Zostało to zaplanowane w celu minimalizacji złożoności projektu oraz zmniejszenia liczby klas, co obniża próg wejścia do kodu aplikacji.

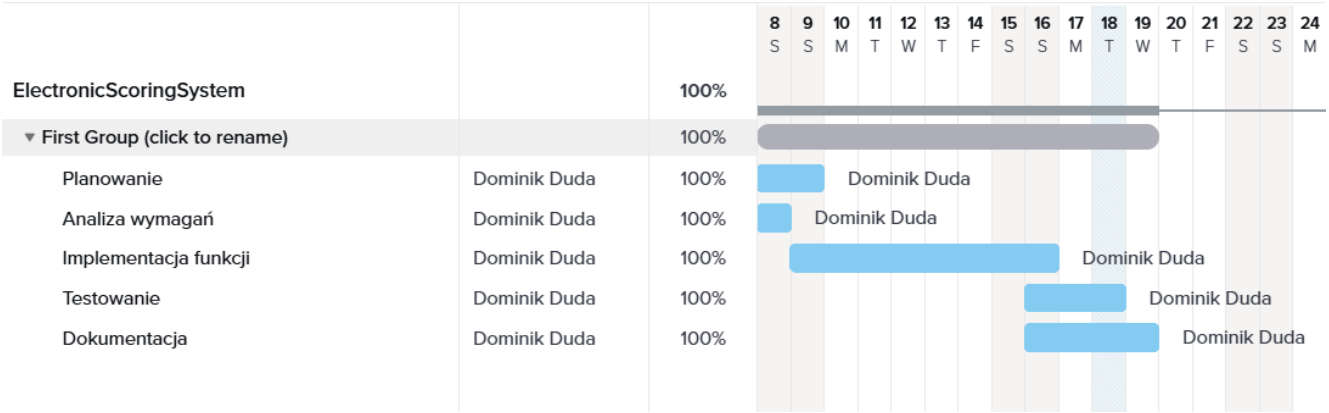
Rozdział 3

Harmonogram realizacji projektu

W niniejszym rozdziale przedstawiono harmonogram realizacji projektu w postaci diagramu Gantta. Diagram ten ilustruje planowane oraz rzeczywiste etapy pracy nad projektem, uwzględniając czas trwania poszczególnych zadań oraz ich zależności.

3.1 Diagram Gantta

Diagram Gantta jest narzędziem służącym do wizualizacji harmonogramu projektu. Przedstawia on zadania w formie poziomych pasków na osi czasu, co pozwala łatwo śledzić postęp prac. W poniższym diagramie zobrazowano kolejne etapy realizacji Elektronicznego Systemu Oceniania.



Rysunek 3.1: Diagram Gantta przedstawiający harmonogram realizacji projektu

3.2 Opis harmonogramu

Realizacja projektu została podzielona na kilka kluczowych etapów:

- Analiza wymagań i planowanie – określenie funkcjonalności oraz wstępne założenia projektu.
- Implementacja – realizacja kolejnych modułów systemu:
  - Stworzenie panelu logowania,
  - Implementacja panelu administratora umożliwiającego zarządzanie użytkownikami,
  - Implementacja panelu nauczyciela (tworzenie testów, zarządzanie klasami, przeglądanie wyników uczniów),
  - Implementacja panelu ucznia (przegląd ocen, dostęp do testów).

- Testowanie i optymalizacja – sprawdzenie działania systemu, poprawa błędów oraz optymalizacja kodu.
- Dokumentacja i finalizacja – przygotowanie dokumentacji projektu oraz jego wdrożenie.

3.3 Problemy napotkane podczas realizacji

Podczas realizacji projektu napotkano kilka trudności, w tym:

- Brak zdefiniowanych relacji w bazie danych, co wymusiło stosowanie licznych zapytań `JOIN`,
- Konieczność uproszczenia struktury aplikacji poprzez ograniczenie liczby klas i brak wyodrębnionej warstwy domeny,
- Zarządzanie danymi bezpośrednio przez repozytoria, co w niektórych przypadkach powodowało konieczność dodatkowego refaktoryzowania kodu.

3.4 System kontroli wersji i repozytorium

Projekt był zarządzany przy użyciu systemu kontroli wersji Git, co umożliwiło efektywną organizację pracy oraz śledzenie zmian w kodzie. Repozytorium projektu znajduje się na platformie GitHub. Repozytorium zostało zarządzane w sposób zapewniający wysoką prędkość tworzenia. Całość kodu została utworzona przy nie wielu iteracjach z takiego powodu że nad aplikacją pracuje jedna osoba na jednym urządzeniu. Dodatkowo twórcy zależy nad wysoką efektywnością przez co tworzenie commitów co dodanie funcjonalności jest raczej zbędne i nie potrzebnie generuje dodatkowy czas. Który może być użyty na większe dopracowanie aplikacji.

Link do github : https://github.com/duduspower/ElectronicScoringSystem

Rozdział 4

Opis warstwy użytkowej

4.1 Wprowadzenie

Warstwa użytkowa w systemie odpowiada za interakcję użytkownika z aplikacją. Obejmuje interfejsy użytkownika dla różnych typów użytkowników, takich jak studenci, nauczyciele oraz administratorzy. Warstwa ta zapewnia możliwość logowania, zarządzania danymi oraz wykonywania operacji na testach i klasach.

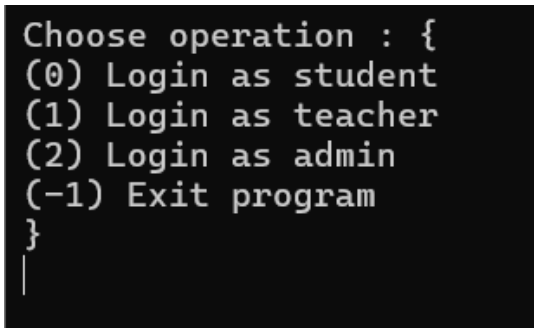
4.2 Główne komponenty warstwy użytkowej

Warstwa użytkowa składa się z kilku kluczowych komponentów, które są odpowiedzialne za obsługę różnych ról użytkowników:

4.2.1 Interfejs logowania (`LoginInterface`)

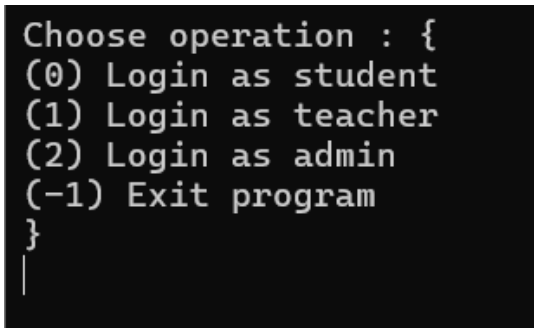
Moduł odpowiedzialny za logowanie użytkowników. Umożliwia wybór roli oraz uwierzytelnianie na podstawie podanych danych dostępowych. Obsługuje następujące operacje:

- Interfejs logowania



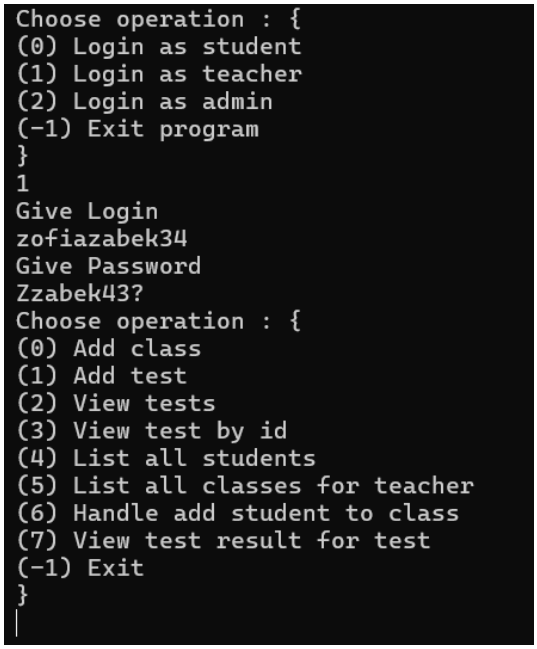
Rysunek 4.1: Interfejs logowania

- Logowanie jako student



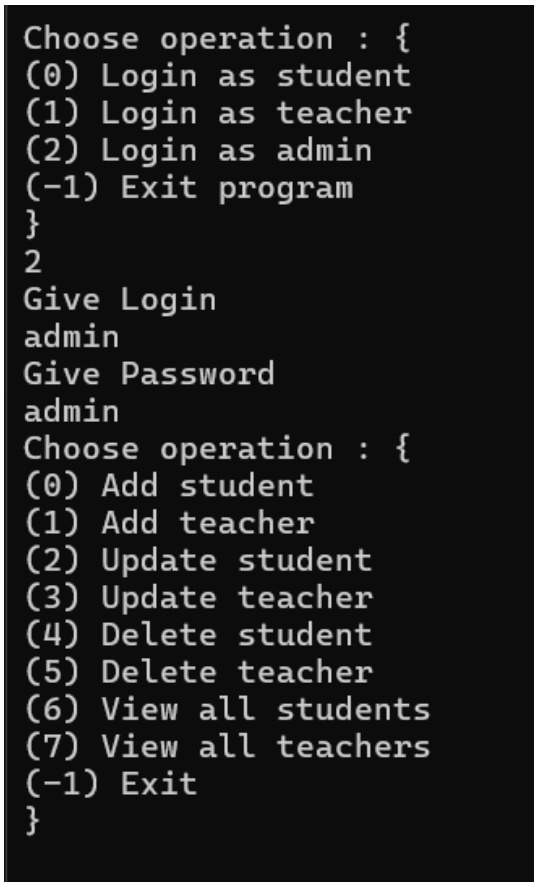
Rysunek 4.2: Logogowanie jako student

- Logowanie jako nauczyciel



Rysunek 4.3: Logowanie jako nauczyciel

- Logowanie jako administrator



Rysunek 4.4: Logowanie jako administrator

4.2.2 Interfejs administratora (`AdminInterface`)

Panel administracyjny umożliwia zarządzanie użytkownikami systemu. Administrator ma dostęp do funkcji takich jak:

- Dodawanie studenta

```
Choose operation : {
(0) Add student
(1) Add teacher
(2) Update student
(3) Update teacher
(4) Delete student
(5) Delete teacher
(6) View all students
(7) View all teachers
(-1) Exit
}
0
Making login :
Give login :
jankowalski12345
Give password :
Jk342!@fda
Making student :
Give name :
Jan
Give surname :
Kowalski
Give date of birth :
2001-05-15
Give email :
jan.kowalski@student.edu.pl
Give phone :
+48123456789
Give index : (letter and 5 numbers)
i12345
Choose operation : {
(0) Add student
(1) Add teacher
(2) Update student
(3) Update teacher
(4) Delete student
(5) Delete teacher
(6) View all students
(7) View all teachers
(-1) Exit
}
6
Student [ID: 1, Name: Dominik, Surname: Duda, Date of Birth: 0001-01-01, Email: dominikduda19@gmail.com
, Phone: +48790896016, Student Index: w69775, Login ID: 4]
Student [ID: 2, Name: Maciej, Surname: Dobrudzki, Date of Birth: 2003-02-17, Email: mdobrudzki42@onet.p
l, Phone: +48532382998, Student Index: i42042, Login ID: 5]
Student [ID: 3, Name: Adam, Surname: Dobek, Date of Birth: 2004-02-15, Email: adobek@gmail.com, Phone:
+48323123142, Student Index: w53422, Login ID: 6]
Student [ID: 1002, Name: Jan, Surname: Kowalski, Date of Birth: 2001-05-15, Email: jan.kowalski@student
.edu.pl, Phone: +48123456789, Student Index: i12345, Login ID: 1002]
Choose operation : {
(0) Add student
(1) Add teacher
(2) Update student
(3) Update teacher
(4) Delete student
```

Rysunek 4.5: Dodawanie studenta

- Edycja studenta

```
(0) Add student
(1) Add teacher
(2) Update student
(3) Update teacher
(4) Delete student
(5) Delete teacher
(6) View all students
(7) View all teachers
(-1) Exit
}
2
Give id of student to update :
1002
Making student :
Give name :
Jan
Give surname :
Nowak
Give date of birth :
2003-04-23
Give email :
jan.nowak@student.edu.pl
Give phone :
+48123456780
Give index : (letter and 5 numbers)
w56565
Choose operation : {
(0) Add student
(1) Add teacher
(2) Update student
(3) Update teacher
(4) Delete student
(5) Delete teacher
(6) View all students
(7) View all teachers
(-1) Exit
}
6
Student [ID: 1, Name: Dominik, Surname: Duda, Date of Birth: 0001-01-01, Email: dominikduda19@gmail.com
, Phone: +48790896016, Student Index: w69775, Login ID: 4]
Student [ID: 2, Name: Maciej, Surname: Dobrudzki, Date of Birth: 2003-02-17, Email: mdobrudzki42@onet.p
l, Phone: +48532382998, Student Index: i42042, Login ID: 5]
Student [ID: 3, Name: Adam, Surname: Dobek, Date of Birth: 2004-02-15, Email: adobek@gmail.com, Phone:
+48323123142, Student Index: w53422, Login ID: 6]
Student [ID: 1002, Name: Jan, Surname: Nowak, Date of Birth: 2003-04-23, Email: jan.nowak@student.edu.p
l, Phone: +48123456780, Student Index: w56565, Login ID: 1002]
Choose operation : {
(0) Add student
(1) Add teacher
(2) Update student
(3) Update teacher
(4) Delete student
(5) Delete teacher
(6) View all students
(7) View all teachers
(-1) Exit
```

Rysunek 4.6: Edycja studenta

- Usunięcie studenta

```
(0) Add student
(1) Add teacher
(2) Update student
(3) Update teacher
(4) Delete student
(5) Delete teacher
(6) View all students
(7) View all teachers
(-1) Exit
}
4
Give id of student to delete
1002
Choose operation : {
(0) Add student
(1) Add teacher
(2) Update student
(3) Update teacher
(4) Delete student
(5) Delete teacher
(6) View all students
(7) View all teachers
(-1) Exit
}
6
Student [ID: 1, Name: Dominik, Surname: Duda, Date of Birth: 0001-01-01, Email: dominikduda19@gmail.com
, Phone: +48790896016, Student Index: w69775, Login ID: 4]
Student [ID: 2, Name: Maciej, Surname: Dobrudzki, Date of Birth: 2003-02-17, Email: mdobrudzki42@onet.p
l, Phone: +48532382998, Student Index: i42042, Login ID: 5]
Student [ID: 3, Name: Adam, Surname: Dobek, Date of Birth: 2004-02-15, Email: adobek@gmail.com, Phone:
+48323123142, Student Index: w53422, Login ID: 6]
Choose operation : {
(0) Add student
(1) Add teacher
(2) Update student
(3) Update teacher
(4) Delete student
(5) Delete teacher
(6) View all students
(7) View all teachers
(-1) Exit
}
```

Rysunek 4.7: Usunięcie studenta

- Dodawanie nauczyciela

```
(0) Add student
(1) Add teacher
(2) Update student
(3) Update teacher
(4) Delete student
(5) Delete teacher
(6) View all students
(7) View all teachers
(-1) Exit
}
1
Making login :
Give login :
anowak
Give password :
Teach3rP@ss!
Making teacher :
Give name :
Anna
Give surname :
Nowak
academicTitle :
Dr inż.
Give email :
anna.nowak@university.edu.pl
Give phone :
+48987654321
Choose operation : {
(0) Add student
(1) Add teacher
(2) Update student
(3) Update teacher
(4) Delete student
(5) Delete teacher
(6) View all students
(7) View all teachers
(-1) Exit
}
7
Teacher [ID: 1, Name: Piotr, Surname: Pietruszka, Academic Title: doktor habilitowany, Email: ppietrusz
ka@wsiz.pl, Phone: +48118224298, Login ID: 2]
Teacher [ID: 2, Name: Zofia, Surname: Ząbek, Academic Title: magister inżynier, Email: zzofia@wsiz.edu.
pl, Phone: +48847737154, Login ID: 3]
Teacher [ID: 1002, Name: Anna, Surname: Nowak, Academic Title: Dr inż., Email: anna.nowak@university.ed
u.pl, Phone: +48987654321, Login ID: 1003]
Choose operation : {
(0) Add student
(1) Add teacher
(2) Update student
(3) Update teacher
(4) Delete student
(5) Delete teacher
(6) View all students
(7) View all teachers
(-1) Exit
}
```

Rysunek 4.8: Dodawanie nauczyciela

- Edycja nauczyciela

```
Choose operation : {
(0) Add student
(1) Add teacher
(2) Update student
(3) Update teacher
(4) Delete student
(5) Delete teacher
(6) View all students
(7) View all teachers
(-1) Exit
}
3
Give id of teacher  to update :
1002
Making teacher :
Give name :
Anna
Give surname :
Nowak
academicTitle :
Mgr inż.
Give email :
anna.nowak@university.edu.pl
Give phone :
+48987654321
Choose operation : {
(0) Add student
(1) Add teacher
(2) Update student
(3) Update teacher
(4) Delete student
(5) Delete teacher
(6) View all students
(7) View all teachers
(-1) Exit
}
7
Teacher [ID: 1, Name: Piotr, Surname: Pietruszka, Academic Title: doktor habilitowany, Email: ppietrusz
ka@wsiz.pl, Phone: +48118224298, Login ID: 2]
Teacher [ID: 2, Name: Zofia, Surname: Ząbek, Academic Title: magister inżynier, Email: zzofia@wsiz.edu.
pl, Phone: +48847737154, Login ID: 3]
Teacher [ID: 1002, Name: Anna, Surname: Nowak, Academic Title: Dr inż., Email: anna.nowak@university.ed
u.pl, Phone: +48987654321, Login ID: 1003]
Choose operation : {
(0) Add student
(1) Add teacher
(2) Update student
(3) Update teacher
(4) Delete student
(5) Delete teacher
(6) View all students
(7) View all teachers
(-1) Exit
}
```

Rysunek 4.9: Edycja nauczyciela

- Usunięcie nauczyciela

```
7
Teacher [ID: 1, Name: Piotr, Surname: Pietruszka, Academic Title: doktor habilitowany, Email: ppietrusz
ka@wsiz.pl, Phone: +48118224298, Login ID: 2]
Teacher [ID: 2, Name: Zofia, Surname: Ząbek, Academic Title: magister inżynier, Email: zzofia@wsiz.edu.
pl, Phone: +48847737154, Login ID: 3]
Teacher [ID: 1002, Name: Anna, Surname: Nowak, Academic Title: Dr inż., Email: anna.nowak@university.ed
u.pl, Phone: +48987654321, Login ID: 1003]
Choose operation : {
(0) Add student
(1) Add teacher
(2) Update student
(3) Update teacher
(4) Delete student
(5) Delete teacher
(6) View all students
(7) View all teachers
(-1) Exit
}
5
Give id of teacher to delete
1002
Choose operation : {
(0) Add student
(1) Add teacher
(2) Update student
(3) Update teacher
(4) Delete student
(5) Delete teacher
(6) View all students
(7) View all teachers
(-1) Exit
}
7
Teacher [ID: 1, Name: Piotr, Surname: Pietruszka, Academic Title: doktor habilitowany, Email: ppietrusz
ka@wsiz.pl, Phone: +48118224298, Login ID: 2]
Teacher [ID: 2, Name: Zofia, Surname: Ząbek, Academic Title: magister inżynier, Email: zzofia@wsiz.edu.
pl, Phone: +48847737154, Login ID: 3]
Choose operation : {
(0) Add student
(1) Add teacher
(2) Update student
(3) Update teacher
(4) Delete student
(5) Delete teacher
(6) View all students
(7) View all teachers
(-1) Exit
}
```

Rysunek 4.10: Usunięcie nauczyciela

- Wyświetlanie listy wszystkich użytkowników

```
Choose operation : {
(0) Add student
(1) Add teacher
(2) Update student
(3) Update teacher
(4) Delete student
(5) Delete teacher
(6) View all students
(7) View all teachers
(-1) Exit
}
6
Student [ID: 1, Name: Dominik, Surname: Duda, Date of Birth: 0001-01-01, Email: dominikduda19@gmail.com
, Phone: +48790896016, Student Index: w69775, Login ID: 4]
Student [ID: 2, Name: Maciej, Surname: Dobrudzki, Date of Birth: 2003-02-17, Email: mdobrudzki42@onet.p
l, Phone: +48532382998, Student Index: i42042, Login ID: 5]
Student [ID: 3, Name: Adam, Surname: Dobek, Date of Birth: 2004-02-15, Email: adobek@gmail.com, Phone:
+48323123142, Student Index: w53422, Login ID: 6]
Choose operation : {
(0) Add student
(1) Add teacher
(2) Update student
(3) Update teacher
(4) Delete student
(5) Delete teacher
(6) View all students
(7) View all teachers
(-1) Exit
}
7
Teacher [ID: 1, Name: Piotr, Surname: Pietruszka, Academic Title: doktor habilitowany, Email: ppietrusz
ka@wsiz.pl, Phone: +48118224298, Login ID: 2]
Teacher [ID: 2, Name: Zofia, Surname: Ząbek, Academic Title: magister inżynier, Email: zzofia@wsiz.edu.
pl, Phone: +48847737154, Login ID: 3]
Choose operation : {
(0) Add student
(1) Add teacher
(2) Update student
(3) Update teacher
(4) Delete student
(5) Delete teacher
(6) View all students
(7) View all teachers
(-1) Exit
}
```

Rysunek 4.11: Wyświetlanie listy wszystkich użytkowników

4.2.3 Interfejs studenta (`StudentUserInterface`)

Interfejs przeznaczony dla studentów, umożliwiający wykonywanie operacji związanych z testami. Student może:

- Przeglądać dostępne testy do wykonania

```
Choose operation : {
(0) Show test to do
(1) Show tests results
(2) Attend test
(-1) Exit
}
0
Test : [ Id : 1, Name : Test z wiedzy ogólnej, Class : Wiedza ogólna GL01]
Test : [ Id : 2, Name : 2 Test z wiedzy ogólnej, Class : Wiedza ogólna GL01]
```

Rysunek 4.12: Przeglądanie dostępnych testów do wykonania

- Sprawdzić wyniki

```
Choose operation : {
(0) Show test to do
(1) Show tests results
(2) Attend test
(-1) Exit
}
1
Result for test: [name : Test z informatyki 4, result : 3, student : Dominik Duda]
```

Rysunek 4.13: Sprawdzenie wyników

- Podejść do testu

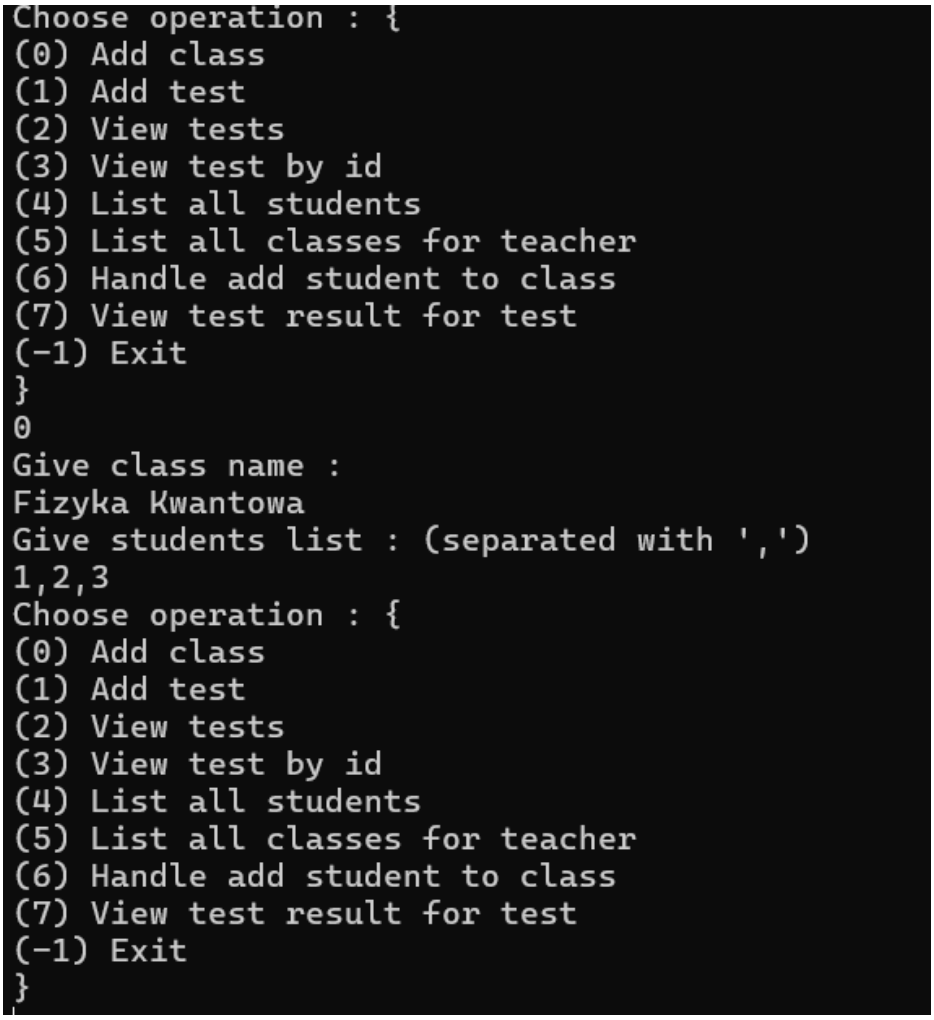
```
Choose operation : {
(0) Show test to do
(1) Show tests results
(2) Attend test
(-1) Exit
}
2
Give id of test to attend :
1006
Attempting test Test z informatyki 4

Która struktura danych działa na zasadzie LIFO (Last In, First Out)?
Answears : {
(0) : Stos (Stack)
(1) : Kolejka (Queue)
(2) : Lista (List)
}
Give answear(numberOfAnswear) :
0

Który z poniższych algorytmów służy do sortowania danych?
Answears : {
(0) : Bubble Sort
(1) : Dijkstra
}
Give answear(numberOfAnswear) :
1

Co oznacza skrót "HTML"?
Answears : {
(0) : Hyper Text Markup Language
(1) : High Technology Modern Language
}
Give answear(numberOfAnswear) :
0
Choose operation : {
(0) Show test to do
(1) Show tests results
(2) Attend test
(-1) Exit
}
```

Rysunek 4.14: Podejście do testu

4.2.4 Interfejs nauczyciela (`TeacherUserInterface`)

Moduł interfejsu nauczyciela pozwala na zarządzanie testami i klasami. Nauczyciel ma możliwość:

- Dodanie klasy



Rysunek 4.15: Dodanie klasy

- Dodanie testu

```
Choose operation : {
(0) Add class
(1) Add test
(2) View tests
(3) View test by id
(4) List all students
(5) List all classes for teacher
(6) Handle add student to class
(7) View test result for test
(-1) Exit
}
1
Making test :
Give test name :
Test z informatyki i programowania
Give class id :
Printing students classes :
Id : 3, Name : Wiedza ogólna GL01
Id : 4, Name : Informatyka GL01
Id : 1002, Name : Fizyka Kwantowa
4
Declare number of questions in test :
2
Making question :
Give question value :
Który protokół jest używany do bezpiecznego przesyłania danych w Internecie poprzez szyfrowanie?
Give number of answears :
4
Give correct answear :
Give answear value
HTTPS
Give answear value
FTP
Give answear value
SMTP
Give answear value
HTTP
Making question :
Give question value :
Która struktura danych działa na zasadzie FIFO (First In, First Out)?
Give number of answears :
4
Give correct answear :
Give answear value
Kolejka (Queue)
Give answear value
Stos (Stack)
Give answear value
Lista jednokierunkowa (Singly Linked List)
Give answear value
Drzewo binarne (Binary Tree)
```

Rysunek 4.16: Dodanie testu

- Podglądnięcie testu

```
Choose operation : {
(0) Add class
(1) Add test
(2) View tests
(3) View test by id
(4) List all students
(5) List all classes for teacher
(6) Handle add student to class
(7) View test result for test
(-1) Exit
}
2
Test : [ Id : 1, Name : Test z wiedzy ogólnej, Class : Wiedza ogólna GL01]
Test : [ Id : 2, Name : 2 Test z wiedzy ogólnej, Class : Wiedza ogólna GL01]
Test : [ Id : 1006, Name : Test z informatyki 4, Class : Informatyka GL01]
```

Rysunek 4.17: Podglądnięcie testu

- Zobaczenie zawartości dodanego testu

```
Result for test: [name : Test z informatyki 4, result : 3, student : Dominik Duda]
Choose operation : {
(0) Add class
(1) Add test
(2) View tests
(3) View test by id
(4) List all students
(5) List all classes for teacher
(6) Handle add student to class
(7) View test result for test
(-1) Exit
}
3
Give testId
1006
Test : [ Id : 1006, Name : Test z informatyki 4, Class : Informatyka GL01]
Question : [ Id : 1014, Name : Która struktura danych działa na zasadzie LIFO (Last In, First Out)?]
Question : [ Id : 1015, Name : Który z poniższych algorytmów służy do sortowania danych?]
Question : [ Id : 1016, Name : Co oznacza skrót "HTML"?]
```

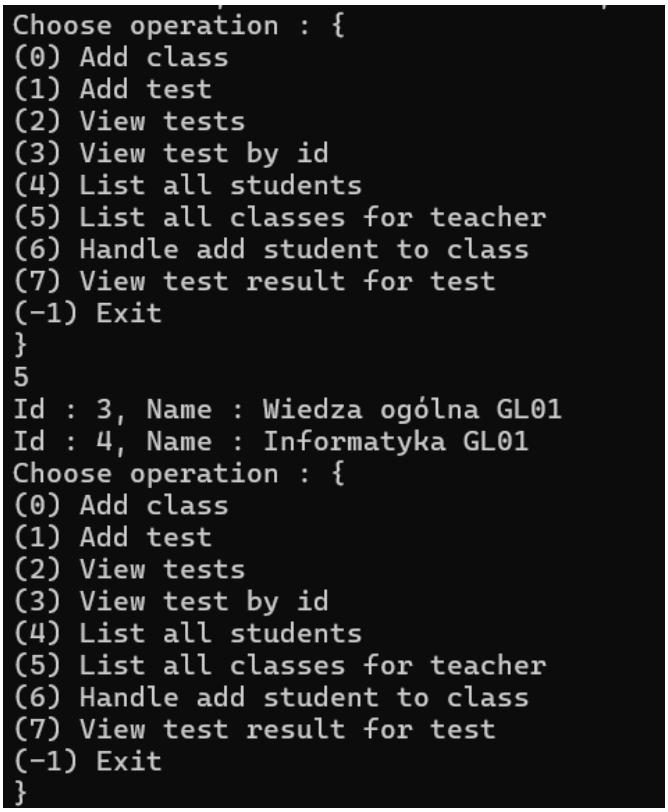
Rysunek 4.18: Zobaczenie zawartości dodanego testu

- Wypisanie wszystkich uczniów

```
Choose operation : {
(0) Add class
(1) Add test
(2) View tests
(3) View test by id
(4) List all students
(5) List all classes for teacher
(6) Handle add student to class
(7) View test result for test
(-1) Exit
}
4
Student [ID: 1, Name: Dominik, Surname: Duda, Date of Birth: 0001-01-01, Email: dominikduda19@gmail.com
, Phone: +48790896016, Student Index: w69775, Login ID: 4]
Student [ID: 2, Name: Maciej, Surname: Dobrudzki, Date of Birth: 2003-02-17, Email: mdobrudzki42@onet.p
l, Phone: +48532382998, Student Index: i42042, Login ID: 5]
Student [ID: 3, Name: Adam, Surname: Dobek, Date of Birth: 2004-02-15, Email: adobek@gmail.com, Phone:
+48323123142, Student Index: w53422, Login ID: 6]
Choose operation : {
(0) Add class
(1) Add test
(2) View tests
(3) View test by id
(4) List all students
(5) List all classes for teacher
(6) Handle add student to class
(7) View test result for test
(-1) Exit
}
```

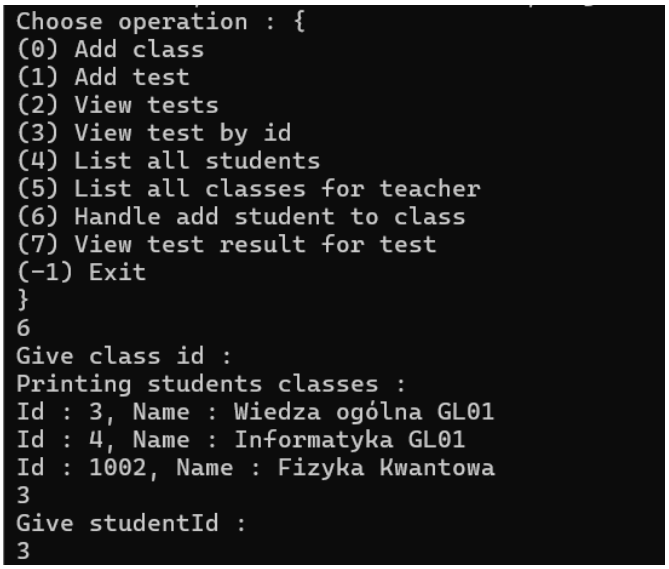
Rysunek 4.19: Wypisanie wszystkich uczniów

- Wypisanie wszystkich klas dodanych przez nauczyciela



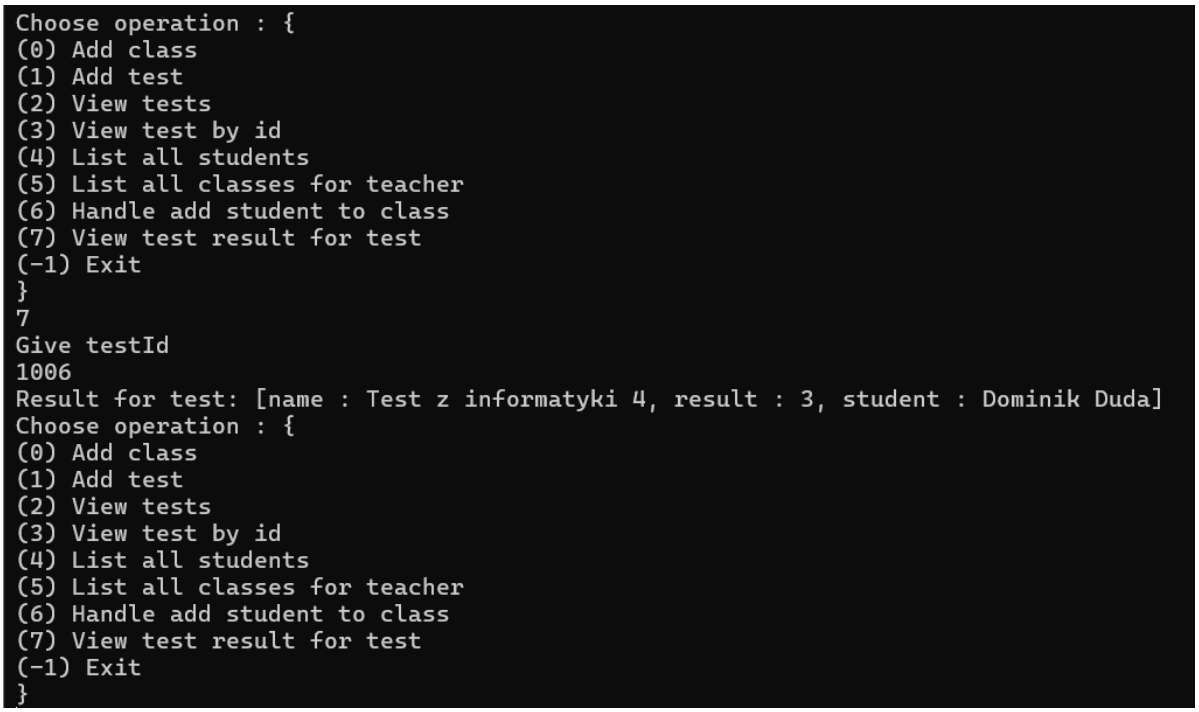
Rysunek 4.20: Wypisanie wszystkich klas dodanych przez nauczyciela

- Dodanie studenta do klasy



Rysunek 4.21: Dodanie studenta do klasy

- Wyświetlenie rezultatu testu



Rysunek 4.22: Wyświetlenie rezultatu testu

4.3 Podsumowanie

Warstwa użytkowa aplikacji zapewnia kompleksowe mechanizmy interakcji z systemem. Każdy z modułów odpowiada za obsługę konkretnych ról i operacji, co zapewnia logiczny podział funkcji oraz łatwość obsługi dla użytkowników końcowych.

Rozdział 5

Podsumowanie

W ramach realizacji projektu stworzono Elektroniczny System Oceniania, który umożliwia automatyzację procesu oceniania testów oraz zarządzanie wynikami uczniów. Aplikacja została zaimplementowana w języku C# z wykorzystaniem Microsoft SQL Server do zarządzania bazą danych. Cała logika aplikacji opiera się na wzorcu projektowym strategia, a interfejs użytkownika jest realizowany w formie konsolowej.

5.1 Zrealizowane prace

W trakcie projektu wykonano następujące zadania:

- **Stworzenie panelu logowania**, który umożliwia autoryzację użytkowników,
- **Implementacja panelu administratora**, który pozwala na dodawanie i zarządzanie użytkownikami (uczniami i nauczycielami),
- **Implementacja panelu nauczyciela**, który umożliwia tworzenie klas, generowanie testów oraz przeglądanie wyników uczniów,
- **Implementacja panelu ucznia**, który pozwala na przeglądanie ocen, dostęp do testów oraz ich rozwiązywanie,
- **Zaimplementowanie mechanizmu walidacji danych**, który dba o poprawność wprowadzanych informacji,
- **Zarządzanie danymi poprzez repozytoria**, które odpowiadają za komunikację z bazą danych,
- **Wprowadzenie podstawowej struktury bazy danych**, wykorzystującej schemat `dbo` i operującej na prostych zapytaniach `JOIN`.

5.2 Problemy i ograniczenia

Podczas realizacji projektu napotkano kilka trudności oraz ograniczeń, które wpłynęły na jego ostateczną strukturę:

- **Brak zdefiniowanych relacji w bazie danych**, co wymusiło stosowanie zapytań `JOIN` do łączenia tabel,
- **Brak wyodrębnionej warstwy domeny**, przez co operacje logiczne są realizowane przez repozytoria lub interfejsy użytkownika,
- **Ograniczona liczba klas**, co ułatwia zrozumienie kodu, ale jednocześnie może utrudnić jego dalszą rozbudowę,

- **Minimalizacja złożoności projektu**, co było świadomym wyborem, jednak w przyszłości może wymagać refaktoryzacji,
- **Zarządzanie danymi bezpośrednio w repozytoriach**, co w niektórych przypadkach powoduje konieczność dodatkowej refaktoryzacji kodu.

5.3 Możliwości dalszego rozwoju

Projekt można rozwijać w kilku kluczowych obszarach:

- **Dodanie warstwy domeny**, co poprawiłoby organizację kodu i oddzieliłoby logikę biznesową od warstwy dostępu do danych,
- **Wprowadzenie bardziej zaawansowanego systemu relacji w bazie danych**, aby zminimalizować liczbę operacji `JOIN` i poprawić integralność danych,
- **Refaktoryzacja kodu**, aby zwiększyć jego modularność i ułatwić dalszą rozbudowę,
- **Dodanie interfejsu graficznego**, co poprawiłoby komfort użytkowania aplikacji,
- **Rozszerzenie funkcjonalności dla nauczycieli**, np. poprzez bardziej szczegółowe analizy wyników uczniów,
- **Optymalizacja zapytań do bazy danych**, aby poprawić wydajność aplikacji przy większej liczbie użytkowników.

Projekt stanowi solidną podstawę do dalszego rozwoju i może być rozbudowywany w zależności od potrzeb użytkowników oraz dostępnych zasobów.

Bibliografia

[1] Andrew Stellman, Jennifer Greene, *Head First C#*, Helion, Gliwice 2009

Spis rysunków

Spis tabel